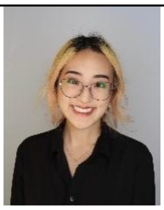
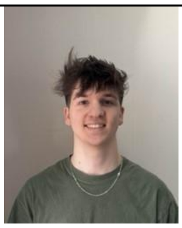
01-T2 - Minecraft Game Mode

Requirements Document (SRS)

CS 4850 – Section 03 – Spring 2025, February 2nd

Professor Sharon Perry

 Ashley Ahn Design	 Matthew Elledge Programmer	AnnaGrace Gwee Design	Bryan Nguyen Team Lead Documentation	Logan Slicker Programmer

Name	Role	Cell Phone	Alternate Email
Ashley Ahn	Design	(404) 953-1062	Ashleyahn.30@gmail.com
Matthew Elledge	Programming	(404) 207-2916	Mtelledge14@gmail.com
AnnaGrace Gwee	Design	(478) 278-0198	Annagracegwee@gmail.com
Bryan Nguyen	Documentation	(470) 237-9923	Bryann0000@yahoo.com
Logan Slicker	Programming	(404) 983-0209	Lslicker24@gmail.com
Sharon Perry	Project Advisor	(470) 555-1212	Sperry46@kennesaw.edu
Norman Reid	Sponsor	(470) 578-5931	Nreid10@kennesaw.edu

Table of Contents

Introduction

Overview

The goal of this project is to complete a functioning game mode for the KSU Minecraft Server. The game mode will be a Tower Defense style game will be coded in Java and should be lightweight and asynchronous for server resources, utilize the Paper API for server tools, have easy to configure settings for the server administrators, and clear documentation for future maintainability of the game mode.

Project Goals

- Milestone 1 – February 10th
    - Working test environment on PC
    - GitHub Repository set up
    - Research on Paper and Plugin Template completed
    - Game Design Document completed
        - Command Structures considered
        - Libraries solidified
        - Game mechanics and rules
        - World and Map Planning

- Milestone 2 – March 17th
    - Verify asynchronous compatibility
    - Working Prototype
    - Completed World/Map
    - Set-up database connection, table creation, and read/write

- Milestone 3 – April 14th
    - Polish Game Mode
    - Bug Testing
    - Play Testing
        - With participation and feedback of members from the server
    - Implement feedback
    - Documentation completed

Definitions & Acronyms

- MC - Minecraft
- API – Application Programming Interface
    - An API is an interface that allows two different types of applications to interface and interact, whether to share, store, or transfer data and information. In this case, we use APIs to interface between Spigot, Paper, and the Minecraft server
- Paper - A type of Minecraft game server designed by the PaperMC organization that is based upon the Spigot architecture, and uses the Paper API to create a more secure, efficient way to modify Minecraft servers
- Spigot – A type of programming interface that is designed to work well with plug-ins for Minecraft servers. The architecture of which Paper is designed and laid on top of

Assumptions

- The developers and players should be capable of running Minecraft
- The players will already have access to the server running the plug-in
- The server will be able to handle game logic and all related processes

Design Constraints

Environment

- Ensure the plug-in effectively use the Paper and Spigot APIs, as well as try to limit the amount of external libraries used for simplicity and minimal overhead
- Using async tasks for saving/keeping less important or time sensitive data will be very necessary to ensure the main processing thread is kept open and unblocked

User Characteristics

- Create necessary levels of moderation, giving admins and moderators more access, with command support, log checking, and force-quitting if necessary, as well as simple end users that might need basic commands that are to be used in game, potentially experience checking or reloading of the GUI in case of a bug.
- GUI should be designed in a logical way to maximize administrative efficiency, as well as help to cultivate and easy learning curve for new and experienced players

System

- Design the events and listeners in ways to reduce computation time by the server to produce lower latency/frame times
- Work alongside any database the server uses to properly save any kind of necessary or wanted player data

Functional Requirements

Map Selection

The game mode should have at least 2 maps at launch, with more to be made. The maps should be distinct in design and gameplay, forcing the player to incorporate how the map is designed in their strategy.

Tower Building

Towers should be distinct in their design and how they function. The player should be made to use more than one type of tower to complete the levels.

Tower Upgrades

Towers should have several upgrade paths that are all variations of the original tower idea, but different enough to have separate use cases depending on the wave, or map design.

Wave Start

Players should receive money upon wave completion and be informed about the next wave to start.

Non-Functional Requirements

Capacity

The game should be able to run on any paper server. It should be lightweight enough to be able to handle higher round difficulties and mob spawning, as well as be able to handle multiple people in each game performing multiple action simultaneously. Lower latency action performance will help to allow a smoother gaming experience if having to make quick, in-the-moment decisions.

Usability

The plug-in should be easy to implement for the developers and any server moderation, and easy to understand after a tutorial or explanation of the ruleset before the game is played. Not only this, but an effortless way to purchase, place, use, and upgrade the towers in the game.

Other

The plug-in will want to ensure it properly masks any admin privileges or commands that should be otherwise inaccessible for security reasons. For any reason, modifications, or further additions of functionality to the plug-in should be no more difficult than adding functionality at the beginning. New mob types, towers, or maps that have unimplemented functionality would be some examples.